Reinforcement Learning: An Introduction

Solutions: Chapter 5

Mrinank Sharma

April 4, 2020

Exercise 5.1: Blackjack

The estimated value is much higher if the player holds 20 or 21 as it is likely that the player wins since the dealer always plays to 17 or higher (or bust), and the winner is whoever holds the higher sum. It is unlikely that the dealer can beat this count.

The estimated value drops off when the dealer is holding an ace because an ace is an advantage because it acts as both 1 and 11, effectively giving the dealer an extra draw.

The frontmost values are higher when the player has a usable ace with sum 12, they are more likely to win. The player effectively gets more draws since if they draw a bad card, they can downgrade the ace to a 1, and this is beneficial for avoiding getting bust which is likely with the current strategy.

Exercise 5.2: First-Visit vs. Every-Visit MC

The results would be the same. It is not possible to a state to be revisited without a usable ace - the player sum can only go up and not say the same.

Also, importantly, this game is properly Markov since the cards are drawn with replacement.

Exercise 5.3: Backup Diagram

The backup diagram is the same as the backup diagram for estimating values except that now there is a value for each state-action pair.

Exercise 5.4: MCES

Instead use arrays $\texttt{counts}[s,a] \leftarrow 0, \quad \forall s,a$ and $Q[s,a] \leftarrow 0, \quad \forall s,a$. Update q using the incremental formula instead of averaging each time, also updating the counts as we go along.

Exercise 5.5: First-Visit vs. Every-Visit MC II

The first visit value is 10 whilst the every-visit value is $10/10 = 1$. Note that the actual value would be $V = 1/(1-p)$.

Exercise 5.6: Action-Value IS

We now also condition on the first action being taken, therefore, the importance weights used start one step later (you could also follow the full derviation of IS to see this). The weighted important sampling estimate becomes:

$$Q(s,a) = \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t+1:T(t)-1}}, \tag{1}$$

i.e., visits are now to state-action pairs.

Exercise 5.7: Weighted Importance Sampling

The weighted importance sampling estimate only includes episodes with behaviour from b which is consisent with π. Our initial estimate of 0 is close to the true value, so we start off performing pretty well. We start to perform worse as more of the runs end up some consisent behaviour, but since the weighted average has very few terms, the variance is high. As we generate more and more consistent episodes, the performing starts to increase again.

Note that the policy is random, so its pretty unlikely to get consisent behaviour. Averaging over 100 runs makes this graph much smoother.

Exercise 5.8: First-Visit vs. Every-Visit MC III

I think that the variance would still be infinite. Let's imagine every-visit MC using standard importance sampling. Going to every-visit effectly adds additional terms to the estimate corresponding to places which weren't first visits.

I'm not fully sure though. Since we are using every-visit, we are no longer summing independent random variables, so we cannot argue that the variance is the sum of individual variances.

Even though we are no longer summing independent random variables (with an episode), there will not be a perfect negative correlation, so the variance (without rescaling) will increse. Therefore, it will remain infinite. We can imagine the worked example summation. Going to every-visit MC will add scaling and more positive terms.

Maybe a nicer way of seeing this. The difference between first-visit and every-visit is the set $\mathcal{T}(s)$. The variance sum now has additional terms in (and a new scaling factor). We already know the original terms sum to infinity. Now, they just sum to a larger amount.

Exercise 5.9: MC policy evaluation: updated

This is a straightforward modification. Modify the `returns` array to store the current estimate and add an additional `counts` array. Replace the update formula with the standard incremental mean update.

Exercise 5.10: Weight-Average Update Rule

The update rule for C_n gives, $C_n = \sum_{k=1}^{n} C_k$.

$$V_n = \frac{\sum_{k=1}^{n-1} W_k G_k}{C_{n-1}} \tag{2}$$

$$\begin{aligned} V_{n+1} &= \frac{\sum_{k=1}^{n-1} W_k G_k + W_n G_n}{C_n} \\ &= V_n + \frac{1}{C_n}[W_n G_n - \underbrace{\sum_{k=1}^{n-1} W_k G_k}_{C_{n-1}V_n} - C_n V_n] \\ &= V_n + \frac{1}{C_n}[W_n G_n - C_{n-1}V_n - C_n V_n] \\ &= V_n + \frac{1}{C_n}[W_n G_n - (C_n - W_n)V_n - C_n V_n] \\ &= V_n + \frac{W_n}{C_n}[G_n - V_n] \end{aligned} \tag{3}$$

Exercise 5.11: Off-policy MC Control

The probability is 1 for the optimal action (according to the current Q). We basically directly substitute this in.

Exercise 5.12: Racetrack Programming

This was quite a painful exercise. I ended up making a number of changes:

- Pessimistically initialise Q. Otherwise, the greedy policy is always a new action. This partially takes into account that it is difficult in general to reach the end.
- I had issues with episodes taking a very long time to finish. This is not surprising - completely random actions are unlikely to reach the end. Combined with the issue of only learning from the tails of episodes, this made learning incredibly slow. I capped the episode length to 50 and gave a large negative reward for reaching this length.
- Learning slowly - I added penalities to hitting the wall in attempt to get things to learn faster. This penalty was the same penalty as the termination penalty (if the penalty is too small, the car will hit the wall in order to try and reach and initialisation state it knows how to finish from, avoiding the termination penalty and getting a better reward).
- I didn't terminate episodes if the wall was hit, though I tried it. The problem with this approach, as I had it implemented, was that if you are going to hit the wall because you are bad a driving, you may as well hit it early and terminate the episode.
- I added a variation on *exploring starts*. Every few episodes, I evaluated the perform of the greedy policy from each starting position. I then picked the starting position of episodes with probaility proportial to how badly we did from that position. Learning tends to be slow anyway, and this helps to speed it up.
- Balancing exploration is hard. If ϵ is too large, random actions are unlikely to lead to our policy changing much. If too small, we never explore. I ended up behaving purely randomly until the performance improved enough/ we found a policy which actually reached the finish line. After that point, I switched to $\epsilon = 0.1$ in an attempt to improve this policy more.
- In this algorithm, the weights grow larger the longer the episodes are (significantly). Therefore, with noise, I think that good actions which were unfortunately were ignored end up performing very poorly and getting overweighted in the importance sampling algorithm. Because of this, I ended up turning off noise. I didn't properly check this hypothesis though. It might have been nice to inspect the IS weights and the episodes they corresponded to.

In general, I'm a bit uncomfortable with the fact that we don't update the importance weights retroactively even though our target policy (the current greedy policy) is constantly changing. I think this non-stationarity doesn't really help us learn much either.

I suppose that I learnt the lesson that MC methods are very very slow! Especially off-policy seems to behave badly.

Regardless, the greedy policies and trajectories can be found in Figures 1 and 2. Most of these policies are pretty reasonable, but its also clear that by inspecting them, they are not optimal (e.g. could cut closer to corners e.t.c.). I ended up doing perhaps 10^5 episodes or so, maybe a bit fewer. My exploring starts variant made this convergence significantly faster, especially for the second track.

Exercise 5.13: Per-Decision IS

$$\rho_{t:T-1}R_{t+1} = \prod_{i=t}^{T-1} \frac{\pi(A_i|S_i)}{b(A_i|S_i)} R_{t+1} \tag{4}$$

We want to show that we can neglect alot of the terms that come up in usual importance sampling. I'll show this for a value estimate, i.e., I'll condition on the state S_t.

$$\begin{aligned}
\mathbb{E}_b[\rho_{t:T-1}R_{t+1}|S_t] &= \mathbb{E}_b\Big[R_{t+1}\prod_{i=t}^{T-1}\frac{\pi(A_i|S_i)}{b(A_i|S_t)}|S_t\Big] \\
&= \int \Big[R_{t+1}\prod_{i=t}^{T-1}\frac{\pi(A_i|S_i)}{b(A_i|S_t)}\Big] p(S_{t+1},R_{t+1}|S_t,A_t)b(A_t|S_t)b(A_{T-1}|S_{T-1})\prod_{i=t+1}^{T-2}p(S_{i+1}|S_i,A_i)b(A_i|S_i)d\{A_i\}_{i=t}^{T-1}d\{S_i\}_{i=t+1}^{T-1} \\
&= \int R_{t+1}\prod_{i=t}^{T-1}\pi(A_i|S_i)p(S_{t+1},R_{t+1}|S_t,A_t)\prod_{i=t+1}^{T-2}p(S_{i+1}|S_i,A_i)b(A_i|S_i)\; d\{A_i\}_{i=t}^{T-1}d\{S_i\}_{i=t+1}^{T-1} \\
&= \int R_{t+1}\pi(A_t|S_t)p(S_{t+1},R_{t+1}|S_t,A_t)\Big[\int\prod_{i=t+1}^{T-1}\pi(A_i|S_i)\prod_{i=t+1}^{T-2}p(S_{i+1}|S_i,A_i)\; d\{A_i\}_{i=t+1}^{T-1}d\{S_i\}_{i=t+2}^{T-1}\Big]dA_t dS_{t+1} \\
&= \int R_{t+1}\pi(A_t|S_t)p(S_{t+1},R_{t+1}|S_t,A_t)dA_t dS_{t+1} \\
&= \int R_{t+1}\frac{\pi(A_t|S_t)}{b(A_t|S_t)}p(R_{t+1}|S_t,A_t)b(A_t|S_t)dA_t \\
&= \mathbb{E}_b[\rho_{t:t}R_{t+1}|S_t].
\end{aligned} \tag{5}$$

This looks far more nasty than it actually is; effectively, write definition, expand out, marginalise to 1 and then you are done.

Exercise 5.14: Discounting Aware IS

This is more involved than it might first look.

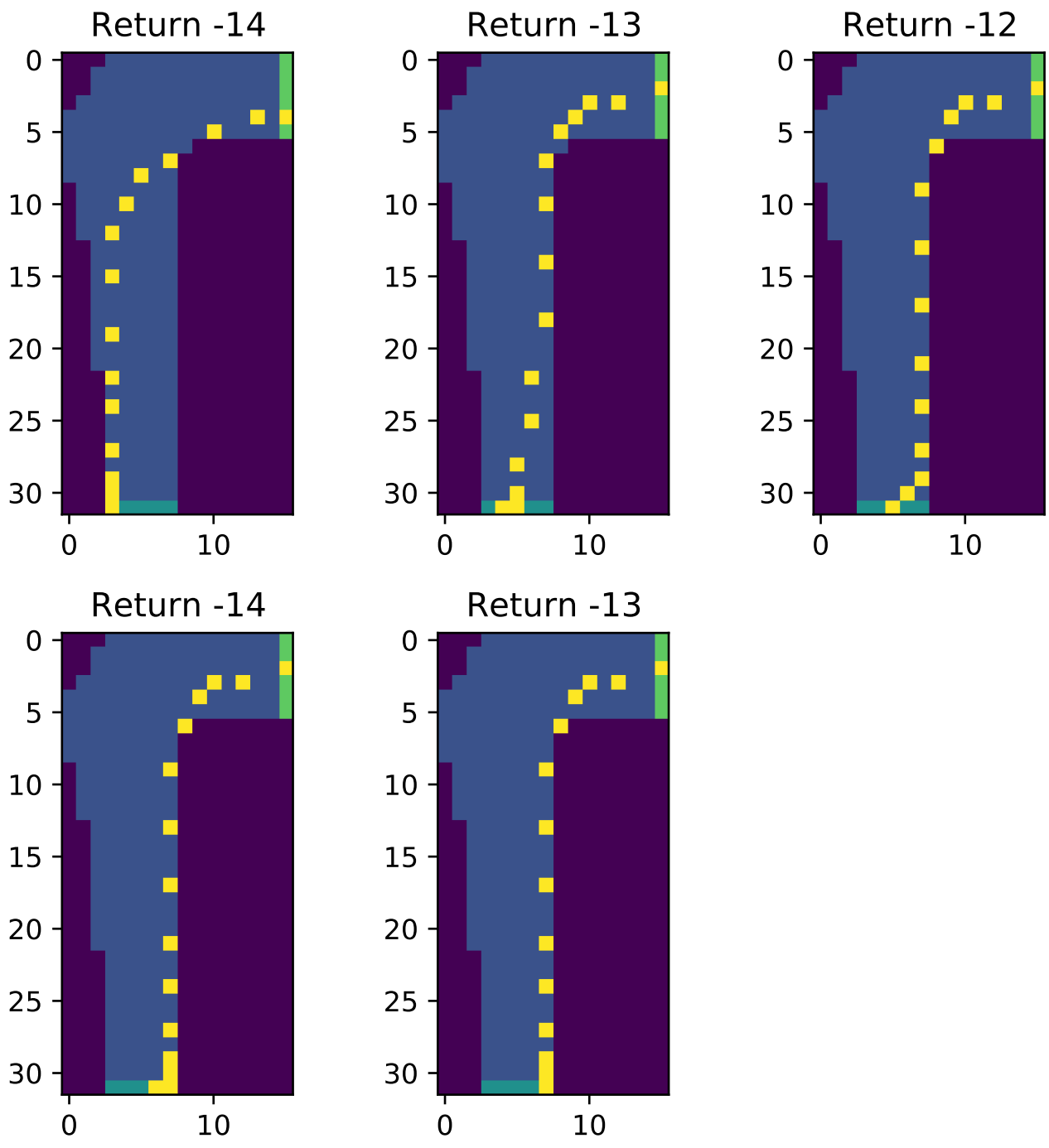

Figure 1: First Track Sample Returns

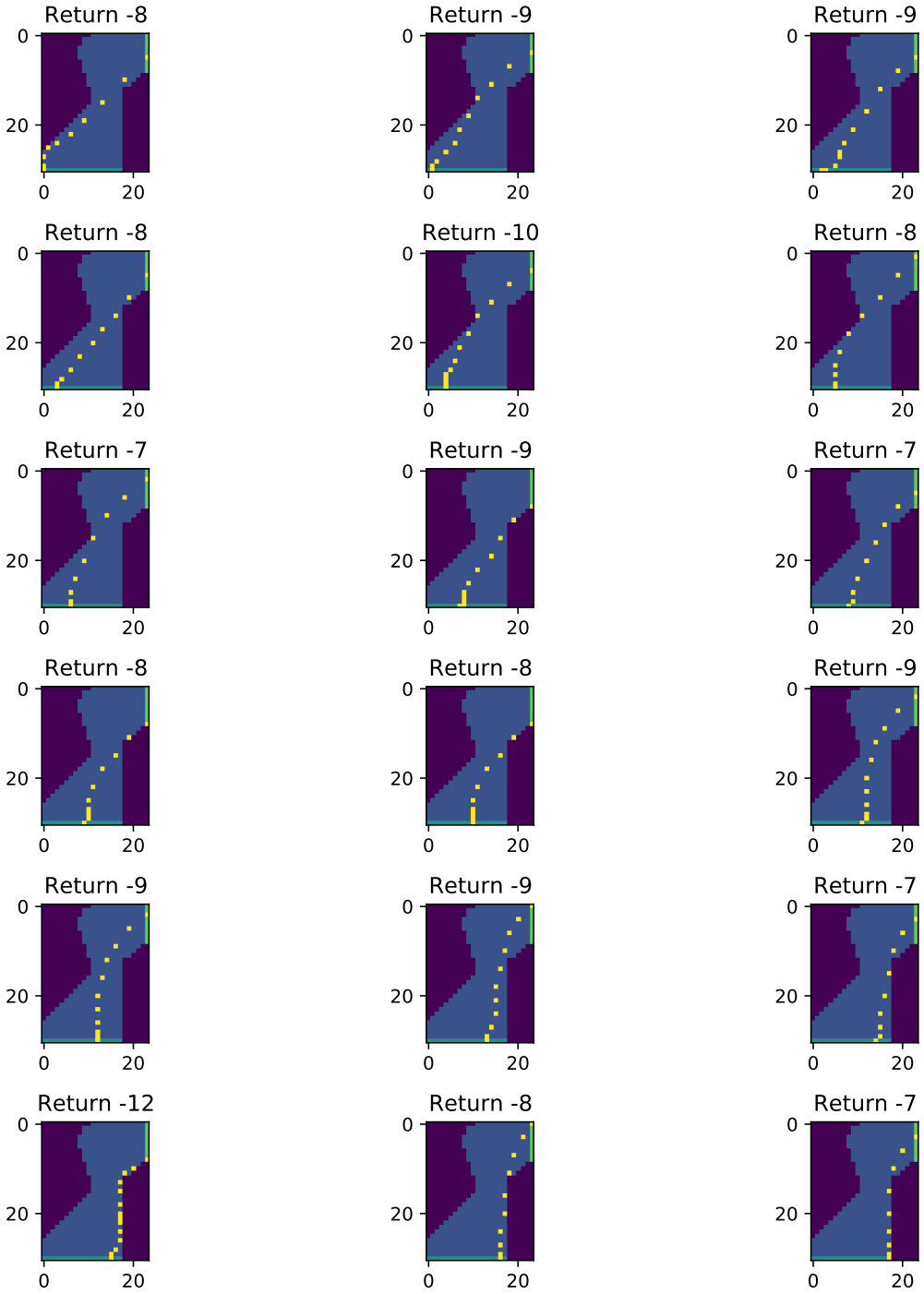

Figure 2: Second Track Sample Returns

Algorithm 1 Off-policy Discount Aware MC Control

Input:

$\gamma \in [0,1]$, discount rate.

Initialisation:

$Q(s,a)$ arbitarily for all $s \in \mathcal{S}, a \in \mathcal{A}$.
$C(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
$\pi(s) \leftarrow \arg\max_a Q(s,a)$, with ties broken **consistently**.

loop:
  $b \leftarrow$ any soft policy
  Generate an episode from under b: $\{S_0, A_0, R_1, S_1, A_1, R_2, ..., S_{T-1}, A_{T-1}, R_T\}$
  for $t \in \{1, 2, \ldots, T-1\}$ **do**: ▷ Proceed forwards rather than backwards
    $\bar{G} \leftarrow 0$
    $W \leftarrow 1$
    $W_h \leftarrow 1$
    $C_h \leftarrow 0$
    for $h \in \{t+1, t+2, \ldots, T\}$ **do**:
      if $A_{h-1} \neq \pi(S_{h-1})$ **then**:
        break
      else:
        $W \leftarrow W \frac{1}{b(A_{h-1}|S_{h-1})}$
      $\bar{G} \leftarrow \bar{G} + R_h$ ▷ Compute Forward Flat Reward
      if $h = T$ **then**:
        $W_h \leftarrow \gamma^{h-t-1} W$
      else:
        $W_h \leftarrow (1-\gamma)\gamma^{h-t-1} W$
      $C(S_t, A_t) \leftarrow C(S_t, A_t) + W_h$
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W_h}{C(S_t, A_t)}[\bar{G} - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ with ties broken consistently.

This algorithm looks very different from the other one. It calculates things forward rather than backwards, so ought to be able to learn well.

If we put $\gamma = 1$ in to this algorithm, it's almost the same, except that now we work forwards rather than backwards. I'm not fully sure, but it seems to work out to basically be the same.